(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 December 2001 (06.12.2001)

PCT

(10) International Publication Number
WO 01/93020 A2

(51) International Patent Classification⁷: G06F 9/00

(21) International Application Number: PCT/IL01/00209

(22) International Filing Date: 5 March 2001 (05.03.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/586,685 1 June 2000 (01.06.2000) US

(71) Applicant *(for all designated States except US)*: **ADUVA INC.** [US/US]; 19731 La Mar Drive, Cupertino, CA 95014 (US).

(72) **Inventors; and**
(75) **Inventors/Applicants** *(for US only)*: **SEGAL, Uri** [IL/IL]; Lipsky Street 16, 62195 Tel Aviv (IL). **TE'ENI, Moddy** [IL/IL]; Wiezel Street 3, 64241 Tel Aviv (IL). **SEGAL, Harel** [IL/IL]; Hanna Snee Street 5, 51586 Bnei Berak (IL).

(74) Agents: **AGMON, Jonathan** et al.; Soroker - Agmon, Advocates and Patent Attorneys, 12th Floor, Levinstein Tower, Petach Tikva Road 23, 66184 Tel Aviv (IL).

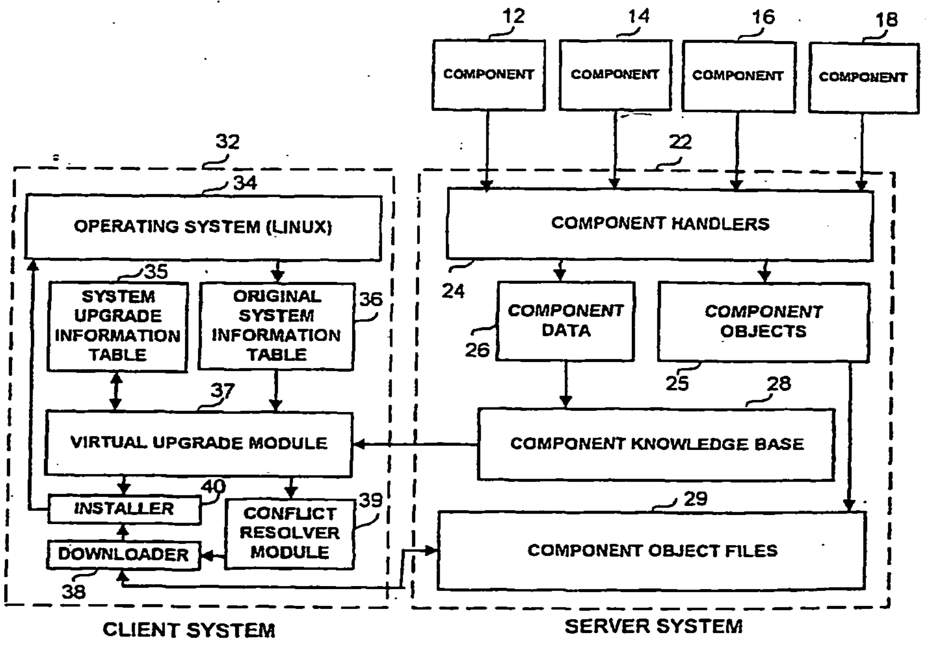
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: VIRTUAL SYSTEM CONFIGURATOR SERVER FOR LINUX



(57) Abstract: A system and method of software system upgrade simulation utilizing a collection of newly developed software modules or new version of existing software modules, information regarding the collection of software modules, and a specific rules language for supporting system managers. The system produces a dependency-conflict-free system configuration file thereby making available the option to system mangers to perform a conflict-free software system upgrade process.

WO 01/93020 A2

A VIRTUAL SYSTEM CONFIGURATOR SERVER FOR LINUX

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

The present invention relates to an apparatus and a method for supporting computer system management, in general, and for providing assistance to system managers in the process of configuring, installing, compiling and upgrading a computer system, in particular.

DISCUSSION OF THE RELATED ART

Computer systems are a collection of computer programs. Modern computer systems are modular in the sense that are designed and actualized using as building blocks a plurality of functionally interconnected software modules. The software modules that constitute the computer systems are computer programs as well, communicating in the standard manner of computer programs by passing arguments.

Requirements from modern computer systems are extremely demanding, therefore a contemporary computer system consists of a remarkably complex set of instructions. Apart from the conventional functions routinely expected from an operating system portion of the computer system such as memory management, file control, I/O commands, buffering, process scheduling and the like, computer systems today are additionally required to function in complex operational settings such as a multi-user environment. The support of a plurality of diverse and sophisticated services such as multitasking and communications became a standard requirement and so did the demand to support a vast and steadily growing number of sophisticated interfacing methods to increasingly advanced hardware devices. As a result prevalent operating systems

Typically, upgrading a computer system involves either adding new components to the ones already in place or replacing existing components by recently developed or improved versions thereof. The process involves collecting the appropriate components from one or more sources (commercial software companies, independent developers, distribution sites on the Internet or any combination thereof) and installing the components by means of appropriate utility programs. The installation utilities could be part of the original computer system or could be supplied by the vendor and/or the developer.

The specific operation of altering a software configuration involves installation, uninstallation, replacing of software components or any combination thereof. For the purposes of a clearer description all types of software configuration changes will be referred to as "upgrading" in the text of this document.

The majority of the available installation utilities operate in a basic fashion. The components of a new type to be installed are inserted and added to the computer system whereas new versions of existing component types overwrite the previous versions rendering them thereby inoperative. The computer system configuration files are updated accordingly and the original configuration files are removed. Installation utilities of a more advanced type include some operational safety measures such as storing the deleted components in specific storage areas, saving original configuration files and providing a list of actions performed. Some specialized utilities check for inter-component dependencies and provide warnings about possible incompatibilities between the original system configuration and the set of components about to be installed. In the face of these unfavorable circumstances the utility programs typically display a warning message, discontinue the upgrade process and permit the system manager to solve the problem in a conventional manner. Solving the problem in such a manner may require a extended time-period as it may involve the reading of a large amount of documentation before proceeding, a plurality of requests for software support

SUMMARY OF THE PRESENT INVENTION

One aspect of the present invention regards a central server system on which a software components upgrade simulation process is based. The central server is having a storage device, an I/O device, a communication device, and an operating system. The virtual system upgrade method consists of gathering information related to software components, researching said information, collecting and storing relevant component objects and associated information, encoding the dependency relationships of the collected components into a model of dependency rules, storing said dependency rules, validating said dependency rules and making available to system managers of connectable client system to utilize the collected, stored and encoded information to perform a conflict-free virtual system upgrade configuration and consequently depending on the results a software system upgrade operation.

A second aspect of the present invention regards a virtual system configuration system based on a central server system for the use of system managers of client system connected thereto. The central server system is having a storage device, an I/O device, a communication device, and an operating system. The virtual system configurator consists of a knowledge database, a component object database, a user preferences table, a rules language module, a component input module, a component validation module, a client requests and queries handling module and a knowledge base handler module.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The preferred embodiments of the present invention are in the context of the Linux environment. Linux is a full-featured UNIX-like computer system that was designed to provide personal computer users a free or very low-cost operating system comparable to the traditional and usually more expensive UNIX systems. Linux has a reputation of a very efficient and fast-performing system. Linus Thorvald of the University of Helsinki in Finland has created the Linux kernel, the central part of the operating system. To complete the operating system, Linus Thorvald and other team members made use of system components developed by members of the Free Software Foundation for the GNU project

Unlike traditional proprietary computer systems, Linux is publicly open and extensible by contributors. Because it conforms to the POSIX standard user and programming interfaces, developers can write programs that can be ported to other operating systems. Linux is distributed commercially by a number of companies.

As the source code was made freely available and further development by others was widely encouraged, a massive, worldwide development is pursued today by a great number of companies, programmer teams and individuals independently or in cooperation with others. As a result of this extensive activity, a multiplicity of new drivers, modules, packages, and applications are written and distributed for Linux constantly. The new systems, versions, and components written for Linux are all potentially installable and operative in every Linux installation circumscribed only by the processor type.

Linux is a powerful and popular operating system and it is widely used on personal computers. It will be readily perceived that a proposed solution designed to support the management of computer systems will be particularly suitable for the Linux environment. The present invention overcomes the disadvantages of the prior art by proposing a novel method and a system, which provides an automated or a semi-automated mechanism to the purpose of altering specially designed knowledge base, which is subsequently validated by repeated testing of the component relationships. The research and validation processes are also continuous. Consequently at any given point in time the knowledge base consist of state-of-art component objects and an up-to-date abstract model of all the existing interdependencies thereof. Typically, the component object files include the most recent component objects available and the latest versions of the existing component objects.

Users of client systems operating in a Lunix environment and requiring a system upgrade connect intermittently to the central server-based knowledge base. The server system provides a specially developed virtual system configurator software that enables the users to export the knowledge base from the server, resolve the conflicts resulting of the virtual system upgrade, download and install the pertinent component objects and perform a real system upgrade. Typically, the process is initiated by user request.

It will be easily perceived by one skilled in the art that in an another embodiment of the invention the above-described process could be automatic or semi-automatic according to the user's decision.

Referring to Fig. 1 there is shown constructed and operative in accordance with the preferred embodiment of the present invention the computerized environment in which Virtual System Configurator Server system 22 is operating. Components 12,14,16,18 are retrieved, entered into server system 22, stored on a memory device on server system 22 and processed by component handlers 24. Component objects 25 are stored into component object files 29 and the associated component data 26 is inserted into Component Knowledge Base 28. Component handlers 24 also perform validation procedures on component data 26 stored in knowledge base 28. Using an expressly designed rules language the functional dependencies of the component objects are encoded into an abstract model representing inter-component dependency rules. The encoded rules are inserted into knowledge base 28 creating new entries or updating the suitable

Referring now to Fig. 2 where a more detailed illustration of server system 22 software configuration is shown. Server 22 contains a component object and component data input module 63, a component validation module 56, a rules language module 58, the knowledge base 28, the component object files 80, a client requests module 76, a client queries and update module 70, a client preferences module 72, a client preferences table 73, a knowledge base handler module 74, an auto-update module 82, an auto-download module 84 and an auto-notify module 86.

The process of creating and maintaining knowledge base 28 is divided into four distinct but logically related stages or activities: (a) component intelligence gathering, (b) component interdependency research, component object and component data input, and (c) rules validation.

Component intelligence gathering 52 and component research activities are performed off-line by special teams. The component intelligence gathering team 52 handles all the activities relating to the retrieval of the entire set of available and operative Linux components and the entire set of available intelligence thereof. All the existing types of components are collected: hardware, software, kernel, RPM (RedHat Package Management) packages, distributions, patches, utilities, editors, compilers, command interpreters, commercial applications, system tools, servers, network modules, file-systems, games, development tools, graphical tools, programming languages, configuration files, interfaces, browsers and the like.

It will be readily perceived that each and every component could have a number of different versions and releases. For instance, the Vim editor used for text editing in UNIX-compatible systems could have two releases (Vim4 and Vim5) and the Vim5 version could have three versions (Vim5.1, Vim5.2 and Vim5.3). Components could be functionally related, for example in software packages comprising a plurality of software modules. Functionally related preferences module 72 is provided to respond to client systems 32 requests for client preferences table 73 updates. Knowledge base handler 74 is responsive to the client systems 32 requests to transmit all or parts of knowledge base 28 to the respective client systems 32 to be used in a user-initiated client system-based virtual system configuration process.

Auto-Download module 84 automatically downloads relevant component objects from component object files 80 in conjunction with the client preferences stored in client preferences table 73. Auto-notify module 86 responds to specific user preferences stored in client preference table 73 in conjunction with Vtree 60 by transmitting notices to the respective client systems 32 with regard to the availability of new components such as kernel patches.

Virtual System Configurator Server system 22 preferably incorporates all the necessary component objects for a required computer system upgrade in a Linux environment. Server system 32 also preferably incorporates the entire set of the requisite component characteristics or component data for a virtual system configuration process in a Linux environment. Server system 22 further develops and dynamically maintains a set of rules associated with the component objects that provides vital information about component object interdependencies.

It will be appreciated by those skilled in the art that in a different embodiment of the present invention a different mode of operation could be used. The present invention is not limited to the specific mode of operation presented

Referring now to Fig. 3 there is shown constructed and operative in accordance with the preferred embodiment of the present invention, Knowledge Base 28. Knowledge Base 28 is a collection of interlinked data structures designed to provide control information for the virtual system configuration process. Knowledge Base 28 comprises a Vtree (Version Tree) Database 60, an Xor rules table 61, an Add-Remove Rules table 63, a Links Table 64, a Stamp table 67, a Stamp Rules table 69, and a Linux Update Modules (LUM) table 66. The functionalities of the tables are described next.

connection between two component data records. A leaf is a node with no children nodes or descendant nodes.

Returning now to Fig. 3 where a block diagram of knowledge base 28 is shown. A component data record stored in Vtree 60 database consist of the following fields: (1) a node id or component id, (2) a component name, (3) a node type, (4) a father id, (5) a next brother id, (6) an install script, (7) an uninstall script, (8) a version number, (9) a release number, (10) an update module number, (11) an icon pointer, (12) a component name for display, (13) a component description and (14) a stamp index. The contents of the fields are described next.

Node id (1) is equivalent to component id that uniquely identifies a component object. Component intelligence module 52 of Fig. 2 designates all received components using a hierarchically organized system of numbering used to express the hierarchical relationships between component objects of the same type or subtype. For example, component object "kernel" could be given the number 1000 and component object "kernel patch" which is a subtype of "kernel" could be given the number 1010. Component name (2) is the component object name such as "compiled kernel" or "rpm". Component node type (3) indicates the type of the tree node such as root, leaf, node and element. Father id (4) indicates the node id of the parent node or of the node immediately above the present node in the tree. Next brother (5) indicates the next node in the tree with an identical parent node to the current node. Install script (6) is a pointer to the specific installation script used for the upgrading process of the component object. Uninstall script (7) is a pointer to the specific uninstallation script used in the upgrading of the component object. Update module number (8) points to the appropriate entry in Linux Update Module table 66. Pointer to icon (9) is a pointer to the icon file associated with the component object. Name for display (10) is the name of the component object to be displayed to the users. Description (11) is a text to be displayed to the users. Stamp index (12) is a pointer to the latest stamp put on the record.

The Rules tables, Xor rules table 61 and Add-remove rules table 63 comprise the inter-component dependency rules. The dependency rules records are created utilizing the rules language. Therefore the operation of the rules tables will be described hereunder in association of the rules language and the following drawings.

In the preferred embodiment of the present invention each component data record represents an abstract component type such as "hardware", "software", "kernel", one or more abstract component subtypes such as "isa cards", "pci cards", 'kernel parameters", "kernel patches" or an non-abstract component subtype such as "PATCH-QIC8-TAPE", "CONFIG_IRDA", "EMACS3.1" and the like. The types and subtypes are organized hierarchically and stored into the nodes of a tree-like data structure. The nodes are interconnected by edges linking the nodes in such a manner that the node containing component type "kernel" is connected to a descendant node thereof containing component subtype "kernel_ base" that in turn connected to a descendant node containing component element "2.2.1", to another descendant node containing component element "2.2.2" and still to another descendant node containing component element "2.2.4".

Referring now to Fig. 4 which is a schematic presentation of vtree 60, organized as a tree-like data structure. The root node 110 is located on the top of tree. Root 110 has three descendants or "children"; a node containing hardware component type 111, a node containing kernel component type 112 and a node holding RPM (Red Hat Package Management) component type 113. The tree node containing hardware component type 111 has three children nodes or descendant nodes: a node holding the record ISA cards component subtype 114, a node holding the record of PCI (Peripheral Component Interconnect) cards component subtype 115 and a node holding the record of serial ports component subtype 116. The node holding the record of hardware component type 111 is the

<u><COMPONENT></u> or <u><COMPONENT> AND <COMPONENT SET></u>

Component choice-sets are a group of integers representing component objects connected with the OR logical operator. For example 3450 OR 6700 OR 1201 indicates that one of the component objects included in the component choice set is needed in installing a dependent component. Formally, a component choice-set as an entity of the rule language can be defined as:

<COMPONENT CHOICE>:

<u><COMPONENT></u> or <u><COMPONENT> OR <COMPONENT CHOICE></u>

Rules-sets consist of a component choice-set connected with the AND logical operator to an another rule-set. For example, (2300 OR 2306 OR 2077) AND (900) indicates that in order to install a specific component object successfully, component object 900 and one of the component objects that are the members of the expression in the brackets are needed to be installed as well. Formally, a rule-set as an entity of the rule language can be defined as:

<RULES SET>:

<u><COMPONENT CHOICE></u> or <u><COMPONENT CHIOICE> AND <RULES SET></u>

Need rule-sets consist of a component set connected by the NEED dependency operator to a rule-set. . For example, (2300 AND 2306) NEED (900 OR 901 OR 902) indicates that in order to be installed successfully, component object 2300 and component object 2306 needs component object 900 or 901 or 902 to be installed as well. Formally, a need rule-set as an entity of the rule language can be defined as:

<NEED RULE>:

<u><COMPONENT SET> NEEDS <RULES SET></u>

Xor rule-sets consist of an XOR logical operator and a component choice-set. For example, XOR (950 OR 952 OR 978) indicates that in order to install successfully a specific component object only one of the component objects on which "tkcvs" is depending have to be installed as well. Component research module 54 obtains the following dependency rule in human language:

Tkcvs component object needs tk component object with a version number greater or equal to 8.0 and tkcvs needs tcl component object with a version number greater or equal to 8.0.

The rule in human language is transformed by research module 54 using rule language module 58 of Fig. 2 into the rule language expression:

TKCVS NEEDS {(TK8.0 OR TK8.1 OR TK8.2) AND (TCL8.0 OR TCL8.1 OR TCL8.2)}

The expression is a need-rule comprising a distinct component object identifier, a NEED operator, and a rules-set. The rules-set comprises two component choice-sets connected with the AND operator. The component choice-set comprises of distinct component object identifiers connected by the OR operator. Therefore the expression is consistent with the terminology of the rules language.

The expression is encoded and inserted into add-remove rules table 63 in the format shown in box 101. "Tkcvs" is identified by the number 100 therefore the field node-id is set to the same number."Tk8.0" is identified by the number 80 therefore the field item-id is set to the same number. "Tk8.0" is contained in the first bracket of the expression therefore the field cond-index is set to 1. In the next two records the fields node-id and cond-index are replicated and item-id is set to the value of 81 that is the identifier of component object "tk8.1" and to the value 82 that is the identifier of component object 'tk8.2" respectively. In the fourth, fifth and sixth line the value of node-id is replicated whereas the field cond-index set to the value of 2 to denote the second bracket in the expression. The field item-id is respectively set to 68,69, and 70 that denote the component objects "tcl8.0", "tcl8.1" and "tcl8.2" respectively.

The NEED operator is assumed, as every need-rule set comprises a NEED operator by definition. In box 104 there is shown the detailed underlying logical syntax of the add-remove rules table.

The next three entries belong to the same xor-rule therefore the value of the field cond-index is replicated. 101 representing "glibc6.1", 128 representing "libc2.0", and 129 representing "libc2.1" are inserted into the field item-id of the second, third and fourth entries respectively. The XOR and OR operators are assumed as every xor-rule set comprises a XOR and an OR operator by definition. In box 108 there is shown the detailed underlying logical syntax of the xor-rules table.

Consequently when a virtual installation of a specific component object will be attempted, conflict resolver module 39 of Fig. 1 will interrogate xor rules table 61 as for the presence of the component object in any of the xor rules tables. If found the other component objects associated with the same xor rule are checked for in the original system information table and if any of them found the installation of the specific component object will be aborted.

Persons skilled in the art will appreciate that the present invention is not limited to what has been particularly shown and described hereinabove. Rather the scope of the present invention is defined only by the claims which follow.

of installing, uninstalling or updating component objects on said client systems computing platforms.

2. The method of claim 1 of researching said information further comprising the steps of:

examining said component information;

scanning said component information;

analyzing said component information for interconnectedness, interrelationships, and interdependency rules;

indexing said component information according to component object types, component object version number and inter-component dependency rules;

organizing said component information into meaningful patterns of component data .

3. The method of claim 1 of storing said component data further comprising the steps of:

inserting specific component data according to attributes observed by said researching into a specific component data structure on the storage device of said central server system;

inserting said dependency rules observed by said researching of component information into specific rules tables on said storage device of said central server system;

creating update modules related to specific component objects to be utilized in said computer object installation process and inserting said update modules into specific update module data structures on said storage device of said central server system.

4. The method of claim 1 of encoding the functional dependencies of said component objects further comprising the steps of:

8. The method of claim 1 further comprising requesting notifications from said central server concerning new component objects or new versions of existing component objects by said system managers.

9. The method of claim 1 further comprising requesting automatic downloading of said new component objects or said new versions of existing component objects to said computer platform running said client system by said system managers.

10. A system of virtual system configurator operating in a computing environment on a central server system platform having a storage device, an I/O device, a communication device, an operating system, and virtual system configuration system comprising:

a knowledge database to hold said component data collected from said component data sources and said dependency rules produced by said researching of said component information;

a component object database to hold said component objects collected from said component object sources;

a user preferences table to hold user preferences regarding automatic downloading of said component objects, automatic notifications of said new component objects or said new versions of existing component objects;

a rules language module to handle the encoding and decoding of said component objects dependency rules;

a component object and component data input module to retrieve said component objects and said component information;

a component object validation module to test said abstract model of said component objects relationships and dependencies;

an auto-download module to export said specific component objects to said client systems by said system managers request;

a stamp table to hold a historical list of said time stamps applied to records of said version tree to distinguish between the different versions of records in said version tree;

an update module table to the to an installation script file to be used during said installation, said uninstallation, and said update process of said component objects.

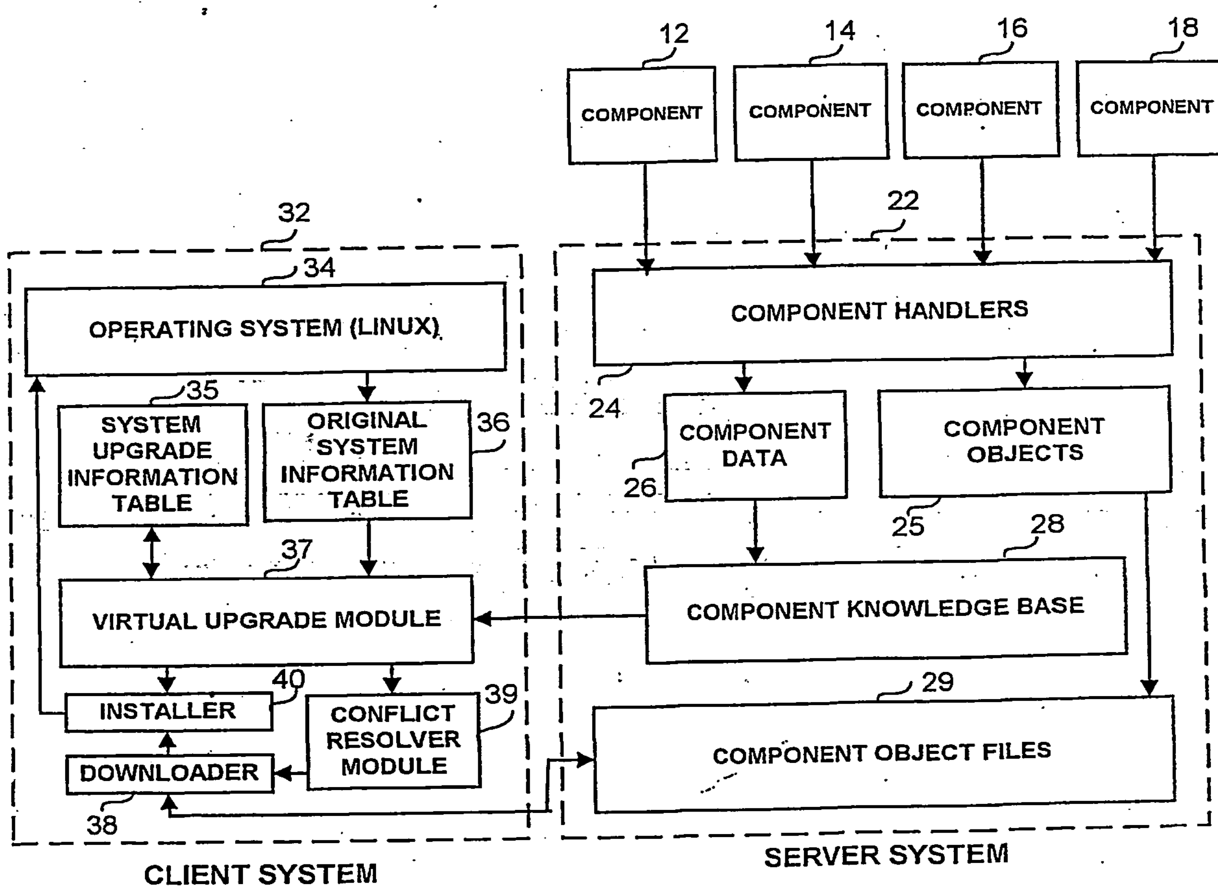
12. The system of claim 10 of said knowledge database further comprising representations of abstract component types to be used as types and subtypes of said component objects.

13. The system of claim 10 of said knowledge database further comprising installable component types data to be used for said installation, said uninstallation and said updating of said component objects.

14. The system of claim 10 of said knowledge database further comprising of useful data fields like a pointer to said installation script type, a version number, a release number, a description, and a time stamp.

15. The system of claim 11 of said xor rules table further comprising a rule index to identify the type of said xor rule and component objects identifying indices to point to said component objects data in said version tree.

16. The system of claim 11 of said add-remove rules table comprising a component object identification index to point to said component object data which needs other components to operate correctly in the same environment.



12
14
16
18
COMPONENT
COMPONENT
COMPONENT
COMPONENT
32
22
34
OPERATING SYSTEM (LINUX)
COMPONENT HANDLERS
35
SYSTEM UPGRADE INFORMATION TABLE
ORIGINAL SYSTEM INFORMATION TABLE
36
24
COMPONENT DATA
COMPONENT OBJECTS
26
25
28
37
VIRTUAL UPGRADE MODULE
COMPONENT KNOWLEDGE BASE
40
INSTALLER
CONFLICT RESOLVER MODULE
39
29
DOWNLOADER
COMPONENT OBJECT FILES
38
CLIENT SYSTEM
SERVER SYSTEM

FIG. 1

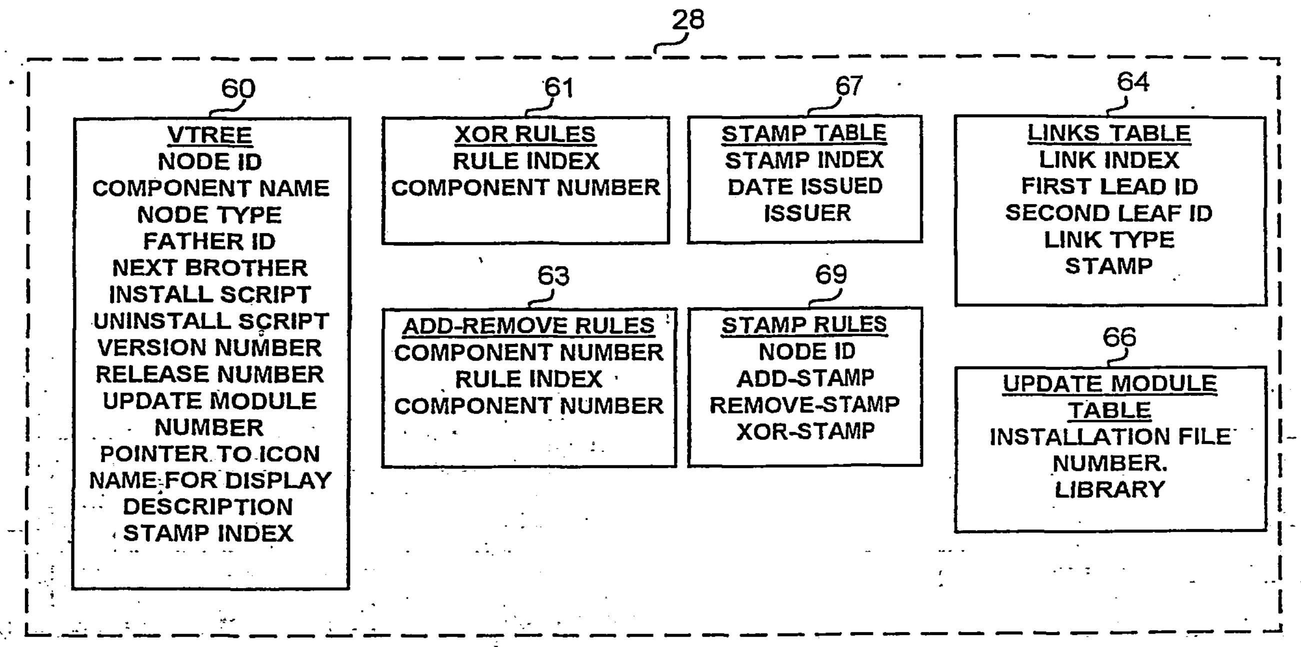
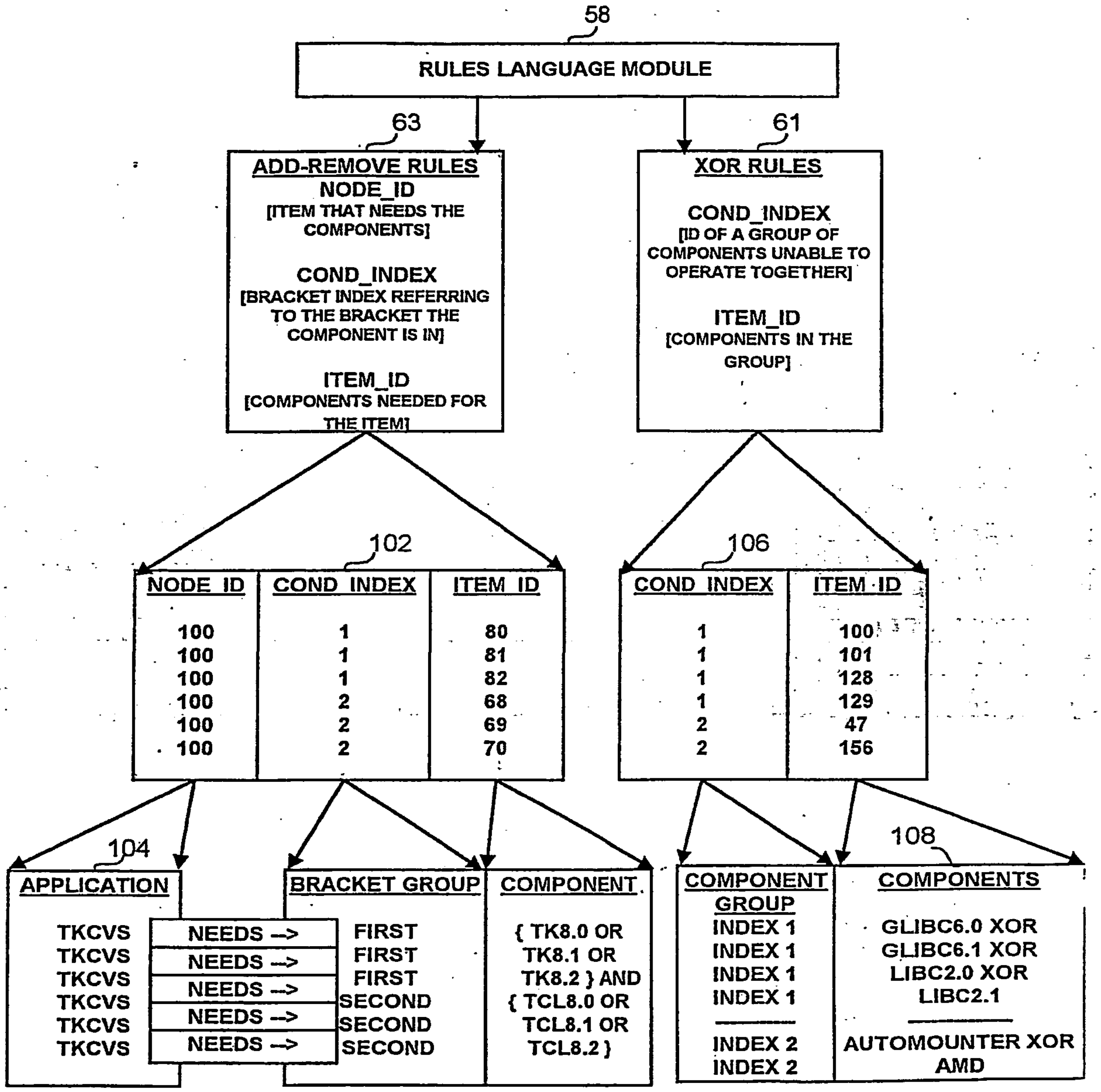

28
60
VTREE
NODE ID
COMPONENT NAME
NODE TYPE
FATHER ID
NEXT BROTHER
INSTALL SCRIPT
UNINSTALL SCRIPT
VERSION NUMBER
RELEASE NUMBER
UPDATE MODULE NUMBER
POINTER TO ICON
NAME FOR DISPLAY
DESCRIPTION
STAMP INDEX
61
XOR RULES
RULE INDEX
COMPONENT NUMBER
67
STAMP TABLE
STAMP INDEX
DATE ISSUED
ISSUER
64
LINKS TABLE
LINK INDEX
FIRST LEAD ID
SECOND LEAF ID
LINK TYPE
STAMP
63
ADD-REMOVE RULES
COMPONENT NUMBER
RULE INDEX
COMPONENT NUMBER
69
STAMP RULES
NODE ID
ADD-STAMP
REMOVE-STAMP
XOR-STAMP
66
UPDATE MODULE TABLE
INSTALLATION FILE NUMBER.
LIBRARY

FIG. 3



RULE TABLES

FIG. 5